VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce klíčových slov v odborných článcích

Keywords detection in research papers

2013 Ondřej Blažek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Ondřej Blažek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Detekce klíčových slov v odborných článcích
Keywords Detection in Research Papers

Zásady pro vypracování:

Cílem práce je provedení průzkumu existujících přístupů, návrh a implementace vybrané nebo vlastní metody a aplikačního prostředí pro experimenty.

1. Průzkum a popis existujících přístupů.
2. Návrh a implementace vybrané nebo vlastní metody.
3. Návrh a implementace počítačové aplikace pro provádění experimentů.
4. Návrh, realizace a hodnocení experimentů.

Seznam doporučené odborné literatury:

[1] Jörg Diederich, Wolf-Tilo Balke: The Semantic GrowBag Algorithm: Automatically Deriving Categorization Systems. ECDL 2007:1-13
Další dle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Miloš Kudělka, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 6. května 2013 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2013 .....

Rád bych na tomto místě poděkoval svému vedoucímu bakalářské práce Mgr. Miloši Kudělkovi, Ph.D. za jeho cenné rady a čas, který mi věnoval při vypracovávání této bakalářské práce .

Abstrakt

Předmětem této bakalářské práce je jedna typická úloha vědecké disciplíny zvané dolování z textu (text mining). Konkrétně tedy detekce klíčových slov dokumentů, jenž mohou sloužit například pro rozdělení dokumentů do kategorií.

Teoretická část je rozdělena na dvě části, kde část první je věnována základním pojmům a jejich objasnění v této problematice. Jedná se především o způsob, jak vhodně reprezentovat dokumenty ve vektorovém prostoru. Druhá část se věnuje průzkumu existujících metod pro určení kategorií dokumentů a detekci klíčových slov, na jejichž základě jsou především tyto kategorie sloučeny.

Významnou součástí práce je také vlastní implementace, kde jsou popsány jednotlivé kroky mého postupu. Setkáme se zde například s postupem vytvoření vektoru, jenž bude reprezentovat dokument a shlukováním sady dokumentů do daného počtu kategorií, na základě jejich podobnosti. Toto shlukování slouží jako nástroj pro kategorizaci, ze které následně díky frekvenční analýze, klíčová slova kategorií detekujeme.

Klíčová slova: kategorizace, tématizace, dolování textu, klíčová slova

Abstract

The subject of this thesis is one typical role of a scientific discipline called text mining. Specifically it is a keyword spotting documents, which can be used for example for the distribution of documents into categories. The theoretical part is divided into two parts where the first part is devoted to the basic concepts and explains them in this issue. This is essentially a way to properly represent documents in a vector space.

The second part deals with the exploration of existing methods for determining the categories of documents and keywords detection on the basis of those categories are merged. An important part of the work is its own implementation, which describes the steps of my process. For example we can find here steps to create a vector that will represent the document and clustering a set of documents into a given number of categories, based on their similarity. This clustering is used as a tool for categorization, which subsequently due to frequency analysis, keywords of categories are detected.

Keywords: categorization , thematization , text mining , key words

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

V dnešní době, době obrovského rozmachu internetu vzniká převážná část psaných dokumentů v digitální podobě. Kvůli takto velké digitalizaci dokumentů, kdy se již dnes bez této formy takřka neobejdeme, protože se využívá ve všech oblastech lidského života, byla potřeba vyvinout automatické zpracování ohromného množství různých informací, obsažených v tomto dostupném, většinou nestrukturovaném textu přirozeného jazyka. Tato vědecká disciplína se nazývá dolování z textu a využívá poznatky ze strojového učení, počítačové lingvistiky a blízké vědecké disciplíny, dolování dat.

Jako cíl mé práce jsem si stanovil prozkoumat existující přístupy a vytvořit aplikaci, která bude schopna rozřadit jednotlivé dokumenty do jednotlivých kategorií, kde kritériem pro rozřazení je podobnost dokumentu, která je právě převážně ovlivněna klíčovými slovy v dokumentech. Díky tomuto, budeme schopni kategorizovat dokumenty, které mají podobný obsah, či se obsahově věnují danému tématu. Nad tímto uskupením pak aplikujeme frekvenční analýzu a na jejím základě budeme moci říct, jaká klíčová slova jsou pro jednotlivé kategorie typická.

Bakalářská práce byla rozdělena na čtyři části. Část první se věnuje základním pojmům, jenž jsou důležité pro orientaci v dané problematice. Především, jak reprezentovat dokument vektorem v určitém vektorovém prostoru. Budu se zabývat způsoby výpočtů jednotlivých složek vektoru a také tím, jak lze minimalizovat počet těchto složek. Rovněž bude rozebráno, jakými způsoby lze spočítat podobnost dokumentů navzájem.

Druhou část věnuji průzkumu existujících přístupů pro určení kategorie dokumentů do předem daných kategorií, či shlukování podobných dokumentů do předem daného počtu shluků, kde nám tyto shluky vytvoří kategorie na základě podobnosti dokumentů.

V části třetí, ve které budu popisovat jednotlivé postupy vytváření mé implementace, se dozvíme, jakou metodu pro kategorizaci jsem použil a jak ji lze implementovat. Budu rozebírat výpočty složek vektorů, reprezentujících dokumenty, popisovat jak jsem došel ke snížení jejich počtu převedením všech slov na jeho kořen a odstranění slov nemající informační význam. Na konci této části popíši možnosti nastavení, jenž má aplikace nabízí. Samozřejmostí je také zobrazení výsledku frekvenční analýzy daných shluků (kategorií), pro detekci klíčových slov.

V poslední části provedu několik experimentů, jako je například vnitřní podobnost dokumentů v dané kategorii, pro zpětné posouzení výsledků shlukování, výsledky frekvenčních analýz jednotlivých clusterů (kategorií) a detekci klíčových slov, jaká jsou pro daný cluster typická.

2 Základní pojmy

Pro porozumění později zde popsaných existujících přístupů pro kategorizaci dokumentů, jenž je základním nástrojem pro určení klíčových slov, je nejprve důležité porozumět následujícím základním pojmům a postupům. Základ jsem především čerpal z [6]

2.1 Sémantická analýza

V této práci si pod tímto slovním spojením budu především představovat identifikaci témat(kategorií) digitálních textových dokumentů. Určování podobností dvou a více dokumentů, které by měly být kategoricky podobné a určování jednotlivých zástupců daných témat.

2.2 Vector space model

Jedná se o nejpoužívanější model pro reprezentaci dokumentů v oblasti vyhledávání informací. V tomto prostoru je každý dokument zobrazen jako bod v n-dimenzionálním prostoru. Každá složka (dimenze), tohoto vektoru, reprezentuje nějaké slovo z kolekce všech dokumentů, nad nimiž kategorizaci provádíme. Tyto číselné složky mohou být vypočítány na základě modelů, které níže popíši.[6]

2.2.1 Binární model

Pokud se daná složka vektoru (slovo) v dokumentu vyskytuje, můžeme tuto informaci pro jednoduchost reprezentovat binárně. Tedy „1“ pokud se slovo v dokumentu vyskytlo a „0“ v případě, že se slovo v dokumentu nenachází.

Příklad 2.1
Mějme pro zjednodušení kolekci čtyř dokumetů ve kterých se nalézá pouze jedna věta.

1. the dog looks out.
2. the man looks at the dog.
3. the man left out.
4. the dog went out for the man.

Nyní vybereme všechna jedinečná slova, která se v dokumentech vyskytují a po abecedním seřazení dostaneme tento vektor slov:
[at, dog, for, left, looks, man, out, the, went]

Pro výše dané seřazení slov budou vypadat dokumenty v tomto vektorovém prostoru následovně:

1. $[0, 1, 0, 0, 1, 0, 1, 1, 0]$
2. $[1, 1, 0, 0, 1, 1, 0, 1, 0]$
3. $[0, 0, 0, 1, 0, 1, 1, 1, 0]$

4. $[0, 1, 1, 0, 0, 1, 1, 1, 1]$

■

2.2.2 TF model

TF (term frequency) představuje rozšiřující způsob binární reprezentace, která nese více informací. Používá se tam, kde je binární reprezentace nedostačující. Rozšíření spočívá v tom, že jednotlivá složka vektoru již není reprezentována pouze číslem „0“ či „1“, které označuje zda se slovo v dokumentu vyskytuje či nikoliv, nýbrž nám dává vědět i počet opakování výskytu tohoto slova v dokumentu.

Pokud bychom chtěli použít pouze TF model pro dlouhé dokumenty, musíme předejít nadhodnocování slov, kde v krátkých dokumentech by se slovo vyskytovalo méně, než v dokumentech delších. Tomuto předejdeme normalizací, a to tím způsobem, že vydělíme celkový počet výskytu daného slova celkovým počtem slov v dokumentu.[6, 7]

Příklad 2.2
Pro reprezentaci dokumentů s použitím TF by vektory z předchozího příkladu bez normalizace vypadaly následovně:

1. $[0, 1, 0, 0, 1, 0, 1, 1, 0]$
2. $[1, 1, 0, 0, 1, 1, 0, 2, 0]$
3. $[0, 0, 0, 1, 0, 1, 1, 1, 0]$
4. $[0, 1, 1, 0, 0, 1, 1, 2, 1]$

■

2.2.3 TF-IDF model

Jak je zřejmé z předchozího modelu, má TF sama o sobě nevýhodu v tom, že nastavuje velkou váhu i obecně častým slovům. Tedy slovům, které můžeme řadit především do skupiny předložek, spojek, sponových (být, stát se) a způsobových sloves (v EN want, do).

Tomuto se můžeme právě vyhnout součinem obou metod TF a IDF, kde nám složka IDF reprezentuje „důležitost“ slova napříč všemi dokumenty. Čím méně se slovo v jednotlivých dokumentech vyskytuje, tím víc vzrůstá jeho důležitost a naopak. Klíčová slova jednotlivých dokumentů pak tedy budou mít svou složku vektoru největší. Matematicky je IDF definován jako logaritmus podílu počtu všech dokumentů ku počtu dokumentů, kde se dané slovo vyskytuje. [6]

Jednotlivá složka vektoru v dokumentu $tdidf_{i,j}$ se dle vzorce vypočítá jako součin TF a IDF. TF je definováno jako podíl celkovému počtu výskytu slov $freq_{i,j}$, kde t_i je počet výskytu slova v dokumentu d_j, ku celkovému počtu slov, která se v dokumentu vyskytují. IDF je definován jako log podílů celkového počtu dokumentů, kde hledáme$|D|$ ku počtu všech dokumentů, které obsahují libovolný počet výskytu slov i.[7]

$$tfidf_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}} * log\left(\frac{|D|}{|\{j : t_i \in d_j\}|}\right)$$

Příklad 2.3
Pro reprezentaci dokumentů s předchozího příkladu a použitím TF-IDF by vektory z předchozích příkladů , vypadaly následovně:

1. $[0, \quad 0.16, 0, \quad 0, \quad 0.30, 0, \quad 0.16, 0, 0 \quad]$
2. $[0.60, 0.16, 0, \quad 0, \quad 0.30, 0.16, 0, \quad 0, 0 \quad]$
3. $[0, \quad 0, \quad 0, \quad 0.60, 0, \quad 0.16, 0.16, 0, 0 \quad]$
4. $[0, \quad 0.16, 0.60, 0, \quad 0, \quad 0.16, 0.16, 0, 0.60]$

■

Příklad 2.4
Pro lepší názornost si dejme jednoduchý příklad. Máme databázi o celkovém počtu 1 000 000 dokumentů a dokument o 300 slovech, kde se vyskytuje 12x slovo„mining“. Toto slovo se vyskytuje celkově pouze v tisících dokumentech.

- TF tedy spočítáme jako 12/300 = 0,04. IDF jako $log\left(\frac{1000000}{1000}\right)$ = 3. Součinem těchto čísel získáme výslednou hodnotu: 0,04*3 = 0,12.

■

Pokud si uvědomíme, že logaritmus 1 je roven 0 a že tento případ vzniká právě u již zmiňovaných obecně častých slov, jenž nám nenesou žádnou informaci. Můžeme tyto slova z analýzy vynechat a zmenšit tak n-dimenzionální prostor pro porovnávání dokumentů.

Poznámka 2.1 Postupy pro snížení vektorového prostoru popíši později.

2.3 Porovnávání dokumentů

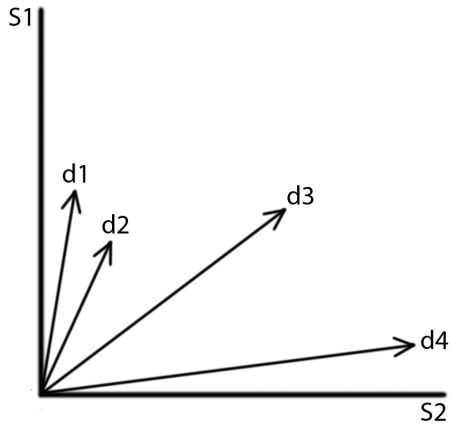
Pokud dokážeme zobrazit do konkrétního vektorového prostoru dokumenty, můžeme je i nyní porovnávat mezi sebou. Ve fulltextových vyhledávačích se toto porovnání používá s dotazem. Pokud hledáme dokumenty vzájemně podobné, hledáme právě takové dokumenty, které mezi sebou mají nejmenší vzdálenost. Nejtypičtějšími metodami pro výpočet vzdálenosti vektoru ve společném vektorovém prostoru jsou:

2.3.1 Euklidovská vzdálenost

Pokud zavedeme v n-rozměrném euklidovském prostoru kartézskou soustavu souřadnic, což se nám hodí obecně pro výpočet a zobrazení vzdálenosti mezi dokumenty reprezentovanými n-dimenzionálními vektory, je vzdálenost d mezi dvěma body X a Z ležícími na souřadnicích $(x_1, x_2, \ldots, x_n), (z_1, z_2, \ldots, z_n)$ určena následujícím vzorcem:

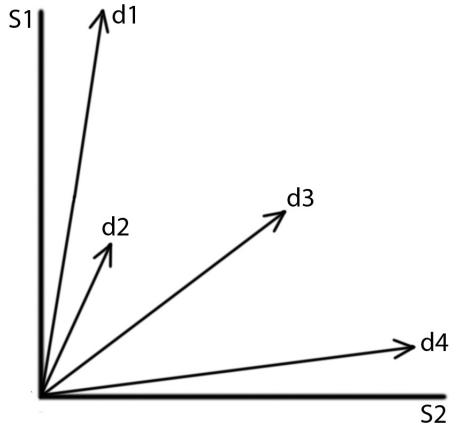
$$d = \sqrt{\sum_{i=1}^{n}(x_i - z_i)^2}$$

Znázornění podobnosti s využitím euklidovské vzdálenosti zobrazující pro jednoduchost čtyři dokumenty ve vektorovém prostoru právě o dvou slovech (dimenzích) S1 a S2 nám zobrazuje *obrázek 1*:



Obrázek 1: Zobrazení čtyř dokumentů ve 2-dimenzionálním prostoru

Vyberme si vektor d2 a hledejme nejbližší jiný dokument z hlediska euklidovské vzdálenosti. Po prvním zhlédnutí je zcela jasné, že nejbližší dokument tomuto, je právě dokument d1. Pokud ovšem vezmeme dokument d1 a všechna jeho slova zkopírujeme a vložíme znova, co se stane v tomto případě?



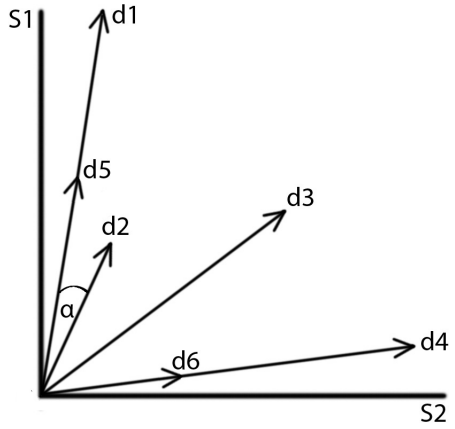
Obrázek 2: Zobrazení čtyř dokumentů ve 2-dimenzionálním prostoru s duplikovaným dokumentem d1

Jak můžeme vidět na *obrázku 2*, euklidovská vzdálenost dokumentu d1 od d2 se zvětšila a nyní je k dokumentu d2 nejpodobnější dokument d3, i když se významově dokument

d1 nezměnil. Tento případ platí při použití TF či TF-IDF modelu. V binární reprezentaci by vektor mohl nabývat pouze tří hodnot (nulu nezapočítáváme). Tomuto problému se vzdáleností vektorů se vyhneme díky normalizací všech vektorů, či použitím kosinové míry místo euklidovské vzdálenosti. [11] [1].

2.3.2 Kosinova podobnost

Určuje míru podobnosti dvou porovnávaných vektorů, jenž se získá výpočtem úhlu, který tyto dva vektory svírají. Jak lze vidět na *obrázku 3*, kde dokumenty $d1$ a $d4$ jsou n-násobkem dokumentů $d5$ a $d6$. Úhel svírající dokumenty $d5$ a $d2$ je stejný, jako u $d1$ a $d2$. Nezáleží tedy vůbec na velikosti vektorů (tj. délce úseček, které je graficky znázorňují).



Obrázek 3: Zobrazení šesti dokumentů s dvěma obsahově stejnými dokumenty.

Je zřejmé, že všechny složky vektoru všech dokumentů budou nezáporná čísla. Tudíž se nacházejí pouze v prvním kvadrantu souřadného systému. Odchylka vektorů A a B se bude tedy pohybovat od 0^o, kdy jsou vektory dokumentu identické, míra podobnosti je tudíž rovna 1.[11]
Pokud budou vektory vzdáleny o 90^o, mají mezi sebou minimální podobnost.Výsledek je tedy 0. Jedná se o velmi přesnou metodu, kde není potřeba vektory normalizovat, pokud nehledíme na výpočetní výkon. Kosinovou podobnost lze tedy pro vektory A a B popsat následujícím vzorcem:

$$podobnost = \cos\alpha = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

Pro výpočet kosinové míry podobnosti počítáme tři skalární součiny, dvě odmocniny a jeden podíl ve zlomku. Pro zrychlení výpočtu, především ve fulltextových vyhledávačích či jen pro optimalizaci řešení, se zde nabízí možnost ukládání si normalizovaných vektorů na velikost 1. Normalizace vektoru se pro zopakování provádí vydělením každé složky

daného vektoru skalárem onoho vektoru, jenž získáme součtem každé složky vektoru umocněné na druhou. Skalár vektoru A se vypočítá:

$$||A|| = \sum_{i=1}^{n} A_i^2$$

Výpočet kosinové míry se díky normalizovaným vektorům redukuje pouze na výpočet jednoho skalárního součinu. Tedy:

$$podobnost = \cos\alpha = \sum_{i=1}^{n} A_i \times B_i$$

2.4 Metody pro snižování počtu dimenzí

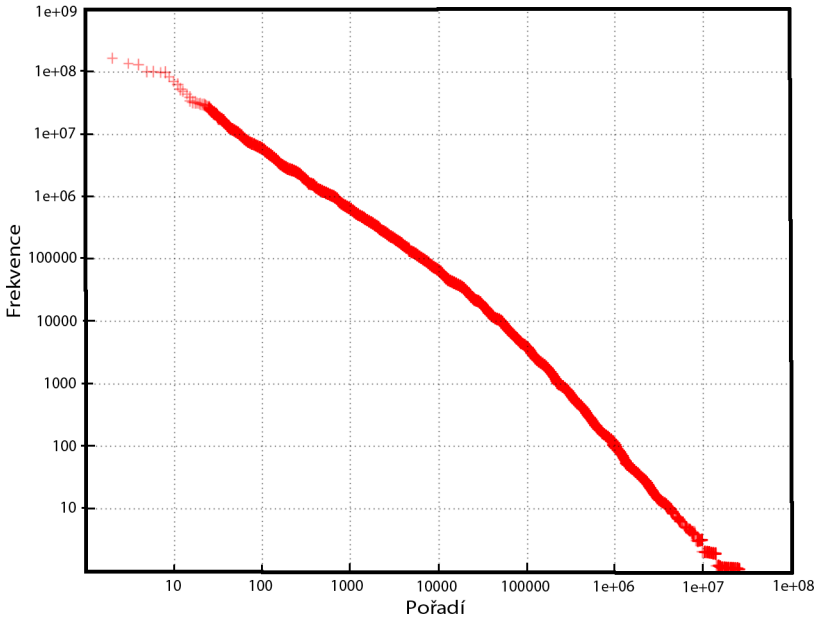
Pokud reprezentujeme dokumenty pomocí vektorového modelu o vysokém počtu dimenzí, kde jak jsme si již dříve řekli, každé složce odpovídá právě jedno slovo, dostaneme se při velkém počtu dokumentů k počtu dimenzí v řádu statisícu. Pokud tedy pro názornou ukázku máme 500 000 dokumentů a ke každému dokumentu vektor o 100 000 dimenzí, velikost uložného prostoru, který potřebujeme pro reprezentaci všech dokumentů, nám vzroste na matici 500 000 x 100 000. Určitě je značně nežádoucí uchovávat matici o celkovém počtu 50 miliard reálných čísel. Tomuto problému se říká takzvané „prokletí dimenzionality“.

2.4.1 Výběr příznaků

Čím víc roste počet dimenzí, tím více exponencinálně roste velikost vektorového prostoru, kde se vektory nacházejí. Při velkém počtu dimenzí, se ve vektorech objevují téměř samé nuly a jednotlivé vektory dokumentů se začínají hromadit na samých okrajích vektorového prostoru. Důsledkem tohoto je vzdálenost mezi dokumenty skoro stejná.

Výběr příznaků nám snižuje dimenzionalitu odstraněním slov, která nemají žádný informační význam. Tato slova nalezneme díky výsledkům frekvenční analýzy. Tedy počtu výskytu všech jedinečných slov v dokumentech. O této frekvenční analýze slov pojednává tzv. **Zipfův zákon** . Tento empirický zákon, resp. zákon statistické povahy, platí ve všech přirozených jazycích. Říká nám, že pokud seřadíme všechna slova z libovolné kolekce textů od největšího výskytu po nejmenší a v tomto pořadí jednotlivá slova očíslujeme od jedné, součin frekvence četnosti slova a jeho číselného pořadí zůstává pro všechna slova skoro konstantní. [18] Pokud je tedy frekvence prvního slova rovna 10 000, druhé nejčastěji vyskytované slovo má frekvenci 5 000, třetí 3 333, čtvrté 2 500 atd. Příklad grafu Zipfova zákona zobrazen výše na *obrázku 4*.

Nejdůležitější skutečností Zipfova zákona je ten, že základ, kteréhokoliv přirozeného jazyka je vytvořen poměrně malou skupinou stále se opakujících slov. Pokud tedy budeme ignorovat ve složkách vektoru dokumentu slova například o menší četnosti než 10, dostaneme se touto redukcí dimenzí vektoru z původního počtu pouze na 15-20%. Ve většině případů se v takto odfiltrovaných slovech jedná o pravopisné či jiné chyby.



Obrázek 4: Příklad grafu zipfova zákona

Tato redukce je tedy velmi velká, jednoduchá a navíc do velké míry odfiltrujeme chyby způsobené lidským faktorem, což je do určité míry taky žádoucí.

Naopak, jak jsem již zmiňoval v kapitole o TF-IDF, kdy u obecně častých slov vycházela složka nulová či nule se dost blížící, můžeme zde vytvořit nějaký seznam stop slov, které jako dimenze ve vektorech reprezentovat nechceme. Jedná se především o předložky a spojky, tedy slova, která nenesou žádnou informační hodnotu. Tato slova jsou ve frekvenční analýze v Zipfově zákoně na prvních místech indexu. Tato část nám nezredukuje počet dimenzí jako předešlá, seznam stop slov je v počtu kolem 330 prvků. O úsporu počtu dimenzí se však jedná.

2.4.2 Extrakce příznaků

Jedná o proces, který snižuje počet dimenzí nahrazením původní množiny, jež tvoří složky vektoru dokumentu, množinou novou. Jedná se především o sjednocení slov, která mají stejný význam, ale jiný tvar. Nejrychlejší způsob jak tohoto dosáhnout je tzv. **lemmatizace** a **stemming**. Obě metody jsou vždy závislé na konkrétním jazyku.[18]

2.4.2.1 Lemmatizace - je operace, která převádí slovo na jeho základní tvar (infinitiv) na tzv. *lemma*, využívá se především u vysoce flektivních jazyků, tedy jazyku, které se vyznačují častým skloňováním, velkým počtem časování, předpon a přípon. Pracující s velkým počtem morfému, což je nejmenší vydělitelná část slova. Toto nám znesnadňuje rozdělit slovo na jednolivé části a určit jeho kořen. Jedná se například o český jazyk.

2.4.2.2 Stemming - je proces, který se běžně používá pro anglický jazyk. Převádí slovo na jeho kořen odstraněním morfologických koncovek a případně i jeho předpon, dle daných souborů pravidel. Základ pravidel tvoří seznam možných koncovek pro odstranění či jejich nahrazení příslušnou koncovkou.

Poznámka 2.2 Stemming podrobněji popíiši v sekci věnující se vlastní implementaci, konkrétně v kapitole převádění na kořeny slov.

2.4.2.3 Nevýhody Největším problémem u obou metod je víceznačnost neboli homonymie, jak už u nepřevedených tak i převedených slov na infinitiv či kořen. U lemmatizace můžeme např. pro slovo „letech“ dostat dvě lemmata a to buď „let“ či „léto“. Tímto se zabývá metoda zvaná **desambiguace**, jenž vybírá správné lemma dle větného kontextu. Stemming se může dopustit chyby například u slov „kníže“ a „knížka“. Obě tato slova mají stejný kořen.[18] Jako příklad pro víceznačné slovo ještě v nepřevedeném tvaru může být třeba slovo „koruna“ (mince, královská koruna, koruna stromu).

Dále můžeme poukázat na slova, jenž nemají stejný základní tvar, avšak mají stejný význam. Tato slova se nazývají synonyma. Pro názornost si udejme slova „vztek“ a „hněv“. Abychom dokázali správně sloučit slova mající podobný či stejný význam, musíme přejít ke složitějším technikám, jako je například latentní sémantická analýza, kterou později popíši.

3 Průzkum a popis existujících přístupů

Techniky porovnávaní dokumentů a rozřazení do nějakých kategorií můžeme rozdělit do dvou základních skupin a to na:

- Učení s učitelem
- Učení bez učitele

Učení s učitelem obecně znamená, že algoritmy musí být vytrénovány pro úkoly jako je klasifikace nad trénovací sadou dokumentů. Tréninkem znamená vycvičit algoritmus na konkrétní vstupy a přiřadit jim konkrétní výstupy(kategorie). Díky nim pak můžeme testovat neznámé vstupy a ty klasifikovat na základě jejich učení. Trénovací sada dokumentů by měla být co nejvíce rozdílná co se týče obsahu, neboť každý dokument nám může reprezentovat právě jednu kategorii

V druhém případě, tedy u algoritmu s učením bez učitele není zapotřebí trénovací sady dat. Jedná se především o shlukovací algoritmy, které nám sdružují obecně data s podobnými vlastnostmi do shluku tzv. clusterů a téma jim přiřazujeme sami nebo nějakými automatickými metodami, jako je výběr nejčastějších slov v daném clusteru či odvození tématu od nejbližších dokumentů v blízkosti těžiště shluku apod. Shluk(cluster) je objekt, který seskupuje data, v našem případě dokumenty, které jsou si navzájem podobné a rozdílné od dokumentů, které patří do jiných shluků. Shlukovací algoritmy mají velmi široké využití jak v informatice, tak v biologii, bio informatice, průzkumu trhu apod.

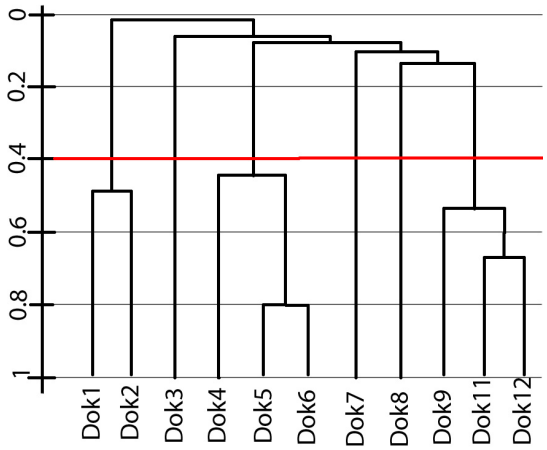
3.1 Hierararchický clustering

Tyto algoritmy spadají do kategorie „učení bez učitele“, kde tedy nemáme trénovací sadu dat, která by nám dala přesné zařazení pod kategorii. Musíme tedy vymyslet způsob přiřazování popisu kategorií automaticky, například dle nejčastěji vyskytnutých slov v daných clusterech. Dělí se na dva následující typy:

3.1.1 Hierarchický aglomerativní clustering (HAC)

Jde o algoritmus, jenž postupuje zdola nahoru. Při počáteční inicializaci je zde každý dokument reprezentován jako jeden shluk(cluster) a pak postupně slučuje dvojice shluků, které jsou nejblíže. Těmto a všem následujícím clusterům, které se skládají alespoň ze dvou dokumentů, se vypočítá střední hodnota(centroid), která reprezentuje daný cluster pro porovnávání. Algoritmus se ukončí až budou sloučeny všechny clustery do jednoho, který obsahuje všechny dokumenty. Tento algoritmus je výpočetně náročný. Při první iteraci je jeho složitost kvadratická, tedy $O(N_2)$, protože musíme vypočítat vzdálenost každého dokumentu s každým.

V každém z následujících kroků se musí přepočítat podobnost vzniklého clusteru ze všemi ostatními clustery, pokud si ukládáme výsledné hodnoty podobnosti z první iterace. Pokud není zachováno provedení výpočetní podobnosti každého clusteru v konstantním čase, zvyšuje se výpočetní náročnost na $O(N_2 \log n)$.



Obrázek 5: Dendrogram HAC 12. dokumentů.

HAC shlukování se typicky zobrazuje jako dendrogram. Na*obrázku 5* vidíme jeho příklad, kde provádíme shlukování na 12. dokumentech. Každá horizontální linie představuje sloučení. První případ sloučení nastal na dok5 a dok6 při podobnosti 0.8 a poslední sloučení na podobnosti kolem 0.02.

Jak je zřejmé, HAC nevyžaduje předem specifikovaný počet clusterů. Pokud ovšem potřebujeme snížit hirearchii na určitém místě, například na podobnosti 0.4, zastavíme algoritmus na dané úrovni podobnosti zkrácením dendrogramu, jak je naznačeno v *obrázku 5*. Zastavením algoritmu dostaneme, jak lze vidět 6 clusterů.
Algoritmus je velice přesný.Vykoupeno je to ovšem výpočetní náročností. Používá se mnohem častěji než divizní clustering.

Existuje mnoho modifikací, pro příklad si uveďme třeba metodu nejbližšího či nejvzdálenějšího souseda, kde se podobnost dvou shluků nepočítá jako vzdálenost mezi těžišti(centroidy), ale nejbližší či nejvzdálenější dvojicí z obou shluků. Další modifikace jsou například w Wardova či Mediánova metoda, viz [4, 17]

3.1.2 Hierarchický divizní clustering

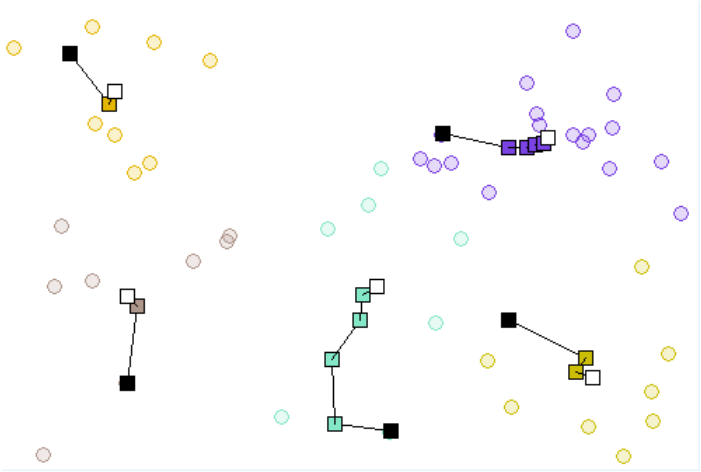
Jde o algoritmus, jenž postupuje zhora dolů. Začíná tedy clusterem, ve kterém jsou všechny dokumenty a končí tehdy, až bude každý dokument reprezentovat jeden cluster. Toto shlukování je koncepčně složitější, protože potřebuje druhý podprogram- flat clustering, jehož úkolem je vytvářet předem daný počet clusterů, které jsou vnitřně koherentní, ale zřetelně odlišné od sebe navzájem. Jinými slovy, aby si dokumenty v rámci clusteru byly co nejvíce podobné a dokumenty, které jsou v jednom clusteru by měly být tak odlišné, jak je to jen možné s dokumenty v jiných clusterech. Nevýhodou tohoto shlukování je časová náročnost nalezení nejlepšího rozkladu množiny o n prvcích na dvě části. Při každém rozkladu tedy musíme prozkoumat $2^{(n-1)} - 1$ možností. Existují implementace, které se snaží snížit složitost na kvadratickou.[5]

3.2 Nehierarchický clustering

Tato shlukovací metoda, jak již název vypovídá je nehierarchická. To znamená, že nevytváří hierarchickou strukturu jako v předchozích případech, ale rozkládá množinu dokumentů na předem zadaný počtů shluků. Po prvním rozkladu se dále nedělí, ale upravuje se tak, aby se optimalizovala odlišnost a vzájemná vzdálenost shluků a aby objekty uvnitř byly rovnoměrně rozděleny. Hlavní myšlenkou je definovat centroidy pro každý cluster, což jsou vlastně těžiště jednotlivých clusterů.

Inicializace počátečních clusterů probíhá v nejjednodušším případě náhodně. Každý jednotlivý cluster se tedy zvolí jako náhodně vybraný dokument. Dalším krokem je, aby se každý dokument přiřadil k nejbližšímu těžišti (centroidu). Pokud jsou všechny dokumenty přiřazeny do nejbližšího clusteru, je první krok dokončen. Následně se přepočítají centroidy všech clusterů, což jsou vlastně průměrné hodnoty složek všech vektorů dokumentů v daném clusteru.

Poté následuje opětovné přiřazení všech dokumentů do nejbližších clusterů. Tento postup se opakuje do doby, dokud není předešlý rozklad roven s aktuálním. Je tedy nalezeno optimální rozložení. Postup je graficky zobrazen pro 50 bodů a pro 5 zvolených clusterů ve 2D na *obrázku 6*, kde prvotní inicializaci centroidů znázorňuje černý čtverec a bílý znázorňuje jeho konečnou polohu. Mezi nimi jsou zobrazena umístění centroidů v jednotlivých iteracích (krocích).



Obrázek 6: Grafické znázornění postupu k dosažení optimálního rozkladu použitím k-means.

Problematice nehierarchického clusteringu se budu věnovat v sekci vlastní implementace.

3.3 Naivní Bayes klasifikátor

Jedná se o jednoduchý pravděpodobnostní způsob kategorizace dokumentů, založený na použití Bayesova teorému s předpoklady nezávislosti mezi prediktory. Lze jej snadno

vytvořit, a to bez složitého opakovaného odhadu parametrů. Toto určitě oceníme u velmi velkých souborů dat. I přes svou jednoduchost tento klasifikátor generuje překvapivě dobré výsledky a je široce používán, protože často překonává složitější metody pro klasifikaci. Jako příklad můžu uvést, že se hojně používá k identifikaci spamu. Jak jsem již uvedl výše, u tohoto způsobu vyžadujeme trénovací množinu. Tedy dokumenty, jejichž kategorie je už určena například znalcem či jiným algoritmem. Na výběru této trénovací množiny závisí i celková přesnost algoritmu, která se může pohybovat až k 90%. [13]

Pokud tedy máme trénovací množinu o dvou dokumentech, kde se vyskytuje celkově jen 5 slov a předpokládáme, že každý tento dokument spadá do jiné kategorie (kat1, kat2), budou nám tedy dokumenty reprezentovat danou kategorii. Viz *-tabulka1.*

dok / slovo	s1	s2	s3	s4	s5
dokument1	7	3	1	0	1
dokument2	1	0	0	7	7

Tabulka 1: Pokusná trénovací množina

Z tabulky 1 vidíme, že pravděpodobnost slova s1, které může být viděno v dokumentu1 je 7/8 a v dokumentu2 1/8. Z tohoto důvodu, jak bude zřejmé níže, bychom neměli všechny pravděpodobnosti výskytu slova nenulové. Například v dokumentu2 by pravděpodobnost výskytu slova s3 byla nulová, tato hodnota by nám tudíž vynulovala součin veškerých hodnot. Tomuto se vyhneme přičtením čísla 1 ke každému slovu v každém dokumentu, což nám nijak významně data neovlivní. Viz *tabulka 2.*

dok / slovo	s1	s2	s3	s4	s5
dokument1	8	4	2	1	2
dokument2	2	1	1	8	8

Tabulka 2: Upravená pokusná trénovací množina

Z tabulky 2 nyní převedeme všechny frekvenční výskyty daných slov na pravděpodobnost výskytu v daných kategoriích a přidáme si zde dva dokumenty, kterým tyto kategorie budeme přidávat .

dok / slovo	s1	s2	s3	s4	s5
kat1	0.8	0.8	0.67	0.11	0.2
kat2	0.2	0.2	0.33	0.89	0.8
dokument3	3	0	1	1	0
dokument4	0	2	0	5	1

Tabulka 3: Kategorie a dva dokumenty k přiřazení

Výpočet výsledné pravděpodobnosti dokumentu d k přiřazení do kategorie k znázorňuje vzorec:

$$P(k|d) = \prod_{1 \leq f \leq n_d} P(s_f|k) \text{ , kde } P(s_f|k) = \frac{P(k|s_f)P(s_f)}{P(k)}$$

Kde $P(s_f|k)$ je podmíněná pravděpodobnost slova s_f vyskytující se v kategorii k. $P(s_f|k)$ interpretujeme jako měřítko toho, jak mnoho náznaků s_f přispívá, že kategorie k má být ta správná. Počet slov, jenž obsahuje průnik slov dokumentu a dané kategorie je n_d. Ze vzorce je vidět, že je nežádoucí, aby pravděpodobnost výskytu v kategorii byla nulová. Celková pravděpodobnost by po součinu byla tudíž nulová také. Bez přičtení jedničky, by nám i pro stejný dokument, který sloužil jako předloha pro kategorii s přidáním byť jednoho slova, které se v ní nevyskytlo, algoritmus přiřadil špatný výsledek. Pro přiřazení kategorie dvou dokumentů v *tabulce 3*, tedy spočítáme pravděpodobnost přiřazení do obou kategorií, jenž je dána podmíněnou pravděpodobností. Pro dokumenty by výpočet tedy vypadal následovně:

$$P(dokument3, kat1) = (0.8^3 \cdot 0.8^0 \cdot 0.67^1 \cdot 0.11^1 \cdot 0.2^0)/kost = 0.03773344$$
$$P(dokument3, kat2) = (0.2^3 \cdot 0.2^0 \cdot 0.33^1 \cdot 0.89^1 \cdot 0.8^0)/kost = 0.00234960$$
$$P(dokument4, kat1) = (0.8^0 \cdot 0.8^2 \cdot 0.67^0 \cdot 0.11^5 \cdot 0.2^1)/kost = 0.00000206$$
$$P(dokument4, kat2) = (0.2^0 \cdot 0.2^2 \cdot 0.33^0 \cdot 0.89^5 \cdot 0.8^1)/kost = 0.01786899$$

Dle výsledků je jasné, že dokument3 by byl přiřazený do kat1 a dokument2 do kat2. Dělení konstantou můžeme pro toto porovnávání vynechat. Jedná se o pravděpodobnost $P(k)$, která nám říká, jakou pravděpodobnost daná kategorie má a že k ní dokument přiřazen bude a tato je pro všechny kategorie stejná.V našem případě 0.5. Dělení jsem si tedy dovolil ignorovat. [10, 12]

3.4 Latentní sémantická analýza (LSA)

I přes postupy pro snižování počtu dimenzí vektoru, zůstává počet dimenzí vysoký. V této metodě dimenze odpovídají spíše tzv. tématům(konceptům, kategoriím), jako jsou třeba filmy, matematika a podobně. Témata se většinou nevybírají ručně, ale automatickými postupy tak, aby co nejlépe odpovídaly tématům v trénovací sadě. Jednotlivá slova nemusí náležet jedné kategorii, ale i jiným, s různými vahami. Díky nutnosti trénovací sady, spadá tato metoda do kategorie „učení s učitelem“. Pro automatické metody identifikace témat je vysoce důležité předem určit jejich počet. Pokud by byly v trénovací sadě obecné texty a témat zvolíme třeba 100, mohou být výsledná témata obecná. Při vyšším počtu se jemnost témat zvýší. Např. místo tématu sport, nám mohou vzniknout témata motor sport, cyklistika, plavání apod.

Tato analýza pracuje i s informacemi o souvýskytu slov. Dokument tedy může být klasifikován do dané kategorie, i když s ní nebude mít společné ani jedno slovo. Pokud se nějaká dvě slova spolu vyskytují častěji než by odpovídalo náhodnému rozložení, budou označena za sémanticky podobná. Předpokládejme, že slova hokej a sport se spolu

budou vyskytovat častěji než hokej a kultura. Toto je souvýskyt první úrovně. Souvýskyt druhé úrovně může byt takový, že slova karate a hokej se spolu v jednom dokumentu nevyskytují, avšak vyskytují se často ze slovem sport. Úrovní souvýskytu může být více, s každou další mírou se však pravděpodobnost souvýskytu snižuje.

Pro nalezení tématu, příslušnosti slov k tématům a míry příslušnosti tématu k dokumentům se používá metoda zvaná singular value decomposition, kde vstupem je pouze kolekce dokumentů a číslo udávající počet témat. Výhodou je, že analýza nám určuje i váhu jednotlivých témat pro celou množinu trénovacích dokumentů. Díky tomu je možné, témata s nejnižší váhou zahodit a snížit tak dimenzionalitu sémantického prostoru.
Témata odpovídají latentní sémantické dimenzi, kde slova sémanticky podobná jsou zobrazována do daného tématu. Témata jsou podána pro člověka obtížně. Pokud však výslednou pravděpodobnost témat pro daný dokument seřadíme od nejvyšší váhy, můžeme odhadnout, čemu přesně téma odpovídá. Z příkladu váhy kategorií níže, můžeme říct, že se v naší lidsky interpretované řeči jedná o téma rally, či automobilové závody.

$$1.7 * zvod, 1.1 * automobil, 0.75 * rally, 0.7 * pou, 0.68 * msto, 0.65 * cesta$$

LSA řeší problémy se synonymy a významově podobnými slovy. Díky nepřesnostem redukcí dimenzionality se setkáváme s žádoucím jevem eliminování chyb v datech. Naopak tato metoda neřeší problém s víceznačností, takže seskupuje slova, která se stejně píšou. Nevýhodou je také vysoká výpočetní náročnost. [8, 9]

Poznámka 3.1 V rámci použití pro získávání informací, se někdy LSA nazývá jako latentní sémantické indexování - LSI.

3.5 Pravděpodobnostní latentní sémantická analýza (pLSA)

Řeší stejný problém jako LSA, ne však na základě lineární algebry, ale je založena na statistice a pravděpodobnosti. Hojně se zde využívá podmíněné pravděpodobnosti, tedy pravděpodobnosti jevu, při určitých omezujících podmínkách. Obecná definice podmíněné pravděpodobnosti je:

$$P(B|A) = \frac{P(A,B)}{P(A)} = \frac{P(A|B)P(B)}{P(A)}$$

Dejme tomu, že v dané množině dokumentů, chceme rozeznat x témat, kde v každém tématu t je spojeno rozdělení pravděpodobnosti přes všechna slova s, které se v dané množině dokumentu vyskytují. Jinými slovy, přiřazuje pro dané téma t pravděpodobnost každému slovu s, díky ní pak můžeme vyjádřit $P(s|t)$ pro jakékoliv slovo a téma. V této analýze se dále předpokládá, že s každým dokumentem D je spojena podmíněná pravděpodobnost výskytu tématu t v tomto dokumentu, tedy $P(t|D)$. Pokud budeme znát předchozí parametry, dokážeme definovat pravděpodobnost výskytu slova s v dokumentu D následovně:

$$P(s|D) = \sum_{t=1}^{x} P(s|t, D) \cdot P(t|D) = \sum_{t=1}^{x} P(s|t) \cdot P(t|D)$$

Jelikož vyžadujeme mít rozdělení pravděpodobností slov pro globální témata, tedy nezávislé na dokumentu, můžeme si dovolit zjednodušení pravděpodobnosti $P(s|t, D)$ na $P(s|t)$. pLSA je navržena tak, že předpokládá nezávislost slov v dokumentu a že každému slovu v dokumentu přiřazuje právě jedno téma. Pravděpodobnost dokumentu vyjádříme jako:

$$P(D) = \prod_{s} P(s|D)^{c(s,D)}$$

$$\log(P(D)) = \sum_{s} c(s, D) \log(P(s|D))$$

$c(s, D)$ udává kolikrát se dané slovo s objevilo v dokumentu D. Jelikož je logaritmus monotónní funkce, zjednodušíme celý výraz zlogaritmováním. Pro určení pravděpodobností $P(t|D)$ a $P(s|t)$, které jsou neznámé, se používá optimalizační úloha, založená na identifikaci takových parametrů obou neznámých pravděpodobností, které maximalizují věrohodnost dat. Tato úloha se řeší např. algoritmem Expectation-Maximization. Po získání pravděpodobnosti $P(t, D)$ pro všechny dokumenty a témata, můžeme libovolný dokument z naší množiny reprezentovat vektorem témat, jako u LSA, kde jak už víme, každá složka určuje pravděpodobnost výskytu tématu v dokumentu.

Výhody tohoto algoritmu jsou v mnohem lepším generování témat než u LSA a jeho matematickém základě. Největší nevýhoda je nemožnost předpovídání témat u nových dokumentů, jenž nebyly přítomny v trénovací množině dokumentů. S rostoucím počtem dokumentů v trénovací množině, roste i počet parametrů, které je třeba odhadnout. To vede k takzvanému přeučování, což je taky velká nevýhoda. [2]

3.6 Latentní Dirichletova alokace (LDA).

Byla poprvé představena v roce 2003, autory jsou David Blei , Andrew Ng , a Michael Jordan.

Jedná se o generativní model, který navíc odstraňuje nevýhody pLSA. Tedy tendence k přeučování díky pevně daným parametrům a nemožnost přiřazení tématu k novým dokumentům. Díky těmto výhodám a vysoké kvalitě výsledků se LDA v praxi používá oproti pLSA téměř všude. Používá dvě pravděpodobnostní rozdělení. Binomické rozdělení nám popisuje četnost výskytu náhodného jevu v n nezávislých pokusech. Pokud neznáme pravděpodobnost tohoto jevu, ale máme o ní částečnou představu, modeluje se její hodnota tzv. beta rozdělením.

Začátek postupu je stejný jako u pLSA, v trénovací množině zvolíme počet témat t. Poté se vyjádří pravděpodobnost generování množiny dokumentů a následně se budou hledat parametry, jenž pravděpodobnost maximalizují. Zde se však nepoužívá pro maximalizaci algoritmus Expectation-Maximalization kvůli složitosti, ale používají se spíše variační metody a Gibbsovo vzorkování. Pro podrobnější popis viz. [15, 16].

4 Návrh a implementace vybrané nebo vlastní metody.

Pro testování mé implementace mi byla poskytnuta sada publikací článku z oboru informatiky. Jednotlivé publikace byly v určitém formátu. Titul byl označen na novém řádku prvotními znaky jako „#*“. Následovaly řádky pro rok vydání, autory apod. Pro nás byl však důležitý abstrakt označený „#!“. Musel jsem si tedy vybrat z dané sady pro náš experiment pouze publikace, které měly titul i abstrakt aby článek měl informační hodnotu. Z celkové sady všech publikací, jsem nakonec přefiltroval okolo 530.000 dokumentů. Sadu dokumentů lze stáhnout z [20]

4.1 Frekvenční analýza

Můj prvotní úkol byl vytvořit frekvenční analýzu výskytu slov, abych viděl, jaká slova se vyskytují nejčastěji. Mohl jsem si tedy ověřit, zda platí Zipfův zákon. Můj první slovník tedy uchovával jednotlivá slova a jejich početní výskyt. Díky této frekvenční analýze jsem si uvědomil, že bude dobré, vytvořit seznam stop slov a filtrovat je, aby se mi ve slovníku nevyskytovala. Stop slova, jak jsem již vysvětloval, nenesou žádnou informační hodnotu. Nejčastější početní výskyt slov ve frekvenční analýze je zobrazen v *tabulce 4*.

Index	Slovo	Frekvence	Součin
1	the	4.557.018	4.557.018
2	of	3.041.797	6.083.594
3	and	2.187.164	6.561.492
4	to	1.804.944	7.219.776
5	in	1.626.617	8.133.085
6	for	1.136.303	6.817.818
7	is	1.128.035	7.896.245
8	we	778.922	6.231.376
9	that	756.420	6.807.780
10	this	704.597	7.045.970

Tabulka 4: Prvních 10 slov ve frekvenční analýze.

Index	Slovo	Frekvence	Součin
17	based	379.540	6.452.180
19	paper	336.928	6.401.632
21	data	297.210	6.241.410
23	system	287.179	6.605.117
26	model	242.255	6.298.630
27	using	233.326	6.229.802
29	time	214.411	6.217.919
30	algorithm	209.441	6.283.230
31	results	192.256.	5.959.936
32	approach	184.504	5.904.128

Tabulka 5: Prvních 10 slov ve frekvenční analýze, bez stop slov.

V *tabulce 5* je zobrazeno prvních 10 slov, která se vyskytují nejčastěji a nejsou stop slova. Jak jsem se přesvědčil, Zipfův zákon platil, tedy samozřejmě s určitou odchylkou, jak lze pozorovat na součinech indexů jednotlivých slov a jejich početního výskytu.

Díky této analýze jsem viděl i chybovost procesu vyhledávání slov, pokud jsem se díval na slova s nejmenším výskytem. Naprostá většina všech slov, která se vyskytla pouze jednou byly způsobené gramatickými chybami či jinými nesrozumitelnými slovy. Přesně jak jsem tedy popisoval u Zipfova zákona. Můžeme tudíž tahle slova, která se nejméně vyskytují, po procesu vyhledávání odstranit. Tuto mez mohu v aplikaci nastavit, defautně jsem použil odstranění slov, která se nevyskytují alespoň 10x. Tato mez ovšem závisí na množství použitých dat.

4.2 Převod na kořeny slov

Pro co nejmenší počet slov, které potřebuji k vytvoření jednotlivých složek vektoru, které budou reprezentovat jednotlivé dokumenty a také kvůli anglicky psané sadě dokumentů, jsem se po průzkumu rozhodl využít stemming. Vytvořil jsem si tedy nový slovník, kde klíč tohoto slovníku bylo slovo, převedené na kořen. Pod tímto klíčem jsem poté přidával původní slova a jejich počet. Díky tomu jsem po převedení a zobrazení libovolného kořene slova mohl vidět všechny jeho původní tvary a dle jejich počtu zjistit, který tvar se vyskytoval nejčastěji.

Pro převod na kořen slova jsem využil porter stemming algoritmus, který je volně šiřitelný pod BSD licencí a po důkladném prozkoumání, jak algoritmus pracuje jsem jej upravil na moje vstupy. Tento algoritmus používá velký soubor definic pro odstranění a nahrazení přípon apod. Základní definice je, že slovo je reprezentováno souhláskami, které si označíme jako C a samohláskami, námi označenými jako V. Pokud se ve slově vyskytuje libovolný počet jdoucích souhlásek nebo samohlásek po sobě (CCC, VVV), tak jsou označeny jen jedním písmenem. Každé slovo nebo jeho část, lze tedy vyjádřit jednou ze čtyř forem:

$CVCV...C$
$CVCV...V$
$VCVC...C$
$VCVC...V$

Toto vše můžeme zkrátit na jednotnou formu:

$$[C]VCVC...[V] \Rightarrow [C]\ (VC)^m\ [V]$$

Hranaté závorky značí libovolnou přítomnost jejich obsahu a $(VC)^m$ nám označuje m-krát opakování VC, kde$m$ je počet slabik ve slově a také se nazývá měřítkem každého slova či jeho části, které je představeno touto formou. Například slova:

m = 0 TRR, EE, A, BEE, TREE
m = 1 TROUBLE, TREES
m = 2 TROUBLES, ORRERY

Díky počtu slabik můžeme kontrolovat měřítko slova. Využívá se hojně tehdy, když chceme zjistit, zda-li je slovo dost dlouhé na to, aby vyhovující část slova pro nějakou operaci(odstranění či změnu) byla posouzena jako přípona a ne jako součást kořene slova. Algoritmus se skládá z šesti kroků redukce, které jsou aplikovány postupně. Celý algoritmus je příliš dlouhý, abych ho zde popisoval. viz. []. Příklad pravidel můžeme vidět v *tabulce 6*.

Pravidlo	Příklad
SSES → SS	caresses → caress
ING →	networking→ network
ATION → ATE	derivation→ derivate
ATE →	derivate→ deriv

Tabulka 6: Příklad pravidel pro převádění na kořeny

Při pohledu na *tabulku 6* si můžeme říct, proč se u slova „derivation“ neodstranila přípona okamžitě, ale až v následujícím kroku, když stejně odstraněna byla. Odpověď na tuto otázku můžeme hledat v šesti krocích, kterými algoritmus postupně prochází. Pravidlo ATION → ATE se nachází v dřívějším kroku jako velké množství jiných pravidel. V pozdějším kroku se pro pravidlo ATE → testuje, zda této příponě nepředchází nějaký znak, konkrétně znak$t$, který by signalizoval, že o koncovku nejde.

Jako příklad uveďme třeba slovo „dotation“. Po prvním kroku vznikne „dotate“, ve druhém kroku se toto slovo nezkrátí, protože příponě předchází písmeno t, v jehož případě se má redukce vynechat. V jiném případě u slova „cation“, se po prvním kroku převede na „cate“ a druhé pravidlo toto slovo ignoruje, neboť jeho měřítko m je malé a jedná se již o kořen a ne příponu.

Dříve jsem uvedl, že v případě převodu na kořeny slov, vznikají chyby při kterých nějaká dvě libovolná významově nepodobná slova mají stejný kořen. Tato chyba nám však vůbec nebo velice málo ovlivní výsledek. Druhá chyba algoritmu je, že převede slovo na špatný kořen. Pro nás je ovšem naprosto irelevantní, jakým tvarem kořene reprezentujeme dané slovo, protože všechna podobná slova se stejným kořenem budou sjednocena stále pod jeden kořen slova, i když neodpovídá realitě.

V mé testovací sadě dokumentů se díky převodu slov na jeho kořen počet redukoval z 82.326 něco málo pod polovinu, přesně na 36.544. Tento počet vyšel při odstranění stop slov a filtrace slov o menším výskytu jak 10.

4.3 Vytváření vektorů dokumentů pomocí TF-IDF

V této fázi jsem si uvědomil, že budu muset vytvořit nový slovník, který by mi reprezentoval číslo dokumentu a nějaký seznam pro reprezentaci všech slov vyskytující se v daném dokumentu. Později jsem tento slovník upravil a místo seznamu jsem použil vniřní slovník pro slovo a ukládání jeho počtu, abych nemusel procházet pokaždé celý seznam a počítat výskyty jednotlivých slov. V tomto vnitřním slovníku je již samozřejmě každé jednotlivé slovo převedeno na kořen.

Další slovník pro urychlení výpočtu IDF jsem si vytvořil takový, kde se ke každému slovu ukládá číselná hodnota která nás informuje o tom, v kolika dokumentech se vyskytuje. Nemusím tedy procházet několikrát celý slovník jednotlivých dokumentů a jejich slo a zjišťovat, zda se dané slovo v dokumentu vyskytlo.

Vzhledem k objemu celkové sady dokumentů, které jsou ukládány ve slovnících a časové náročnosti jsem začal pro následující testování pracovat s menším souborem dokumentů. Pokud si vezmeme, že jedno reálné číslo typu float reprezentuje jednu dimenzi

a zabírá 4 bajty v paměti, vypočteme jednoduchým výpočtem, že pokud se vektor dokumentu rovnal například 10 000, zabíral by v paměti cca 39 kb. Při počtu 530 tisíc dokumentů by jen pro uchování matice všech vektorů bylo zapotřebí přibližně 20Gb.

Počet dimenzí vektoru, reprezentující dokument, je dán počtem všech unikátních kořenů v našem statistickém slovníku. Při výpočtu konkrétní složky vektoru si tedy zkontroluji, kolikrát se dané slovo objevuje v dokumentu a následně pokud mi slovník nějaký záznam vrátil, pokračuji s výpočtem IDF. Dále vezmu celkový počet dokumentů a toto číslo vydělím počtem dokumentů ve, kterých se vyskytlo dané slovo, které se vyskytuje ve zmiňovaném předzpracovaném slovníku. Výsledný podíl zlogaritmuji. Celková hodnota složky vektoru je pak dána součinem TF a IDF.

Všechny slovníky tvořím již za procesu získávání slov z textu. Celkový čas zpracování se přitom díky plnění slovníků nijak závratně nezvětšil. U času pro výpočet vektorů, jsou tyto slovníky poznat mnohonásobně.

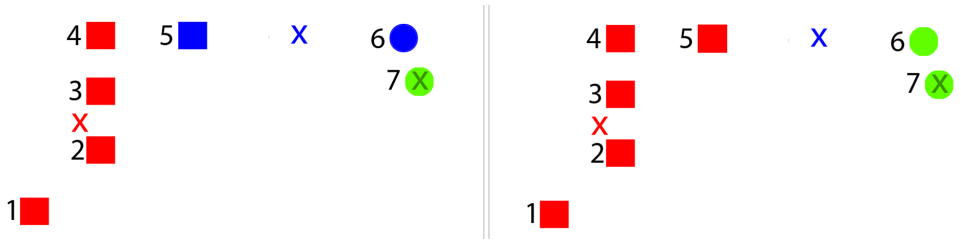
4.4 Shlukovací algoritmus

Pro kategorizaci dokumentů jsem si vybral nehierarchický algoritmus K-means. Zaujal mě především pro své velké použití v praxi. Shlukuje podobné objekty, dle daných kritérií. V našem případě tedy porovnává kosinovou podobností vektory reprezentující dokumenty.

4.4.1 Náhodná inicializace

V první verzi programu jsem si nechával inicializovat počáteční centroidy náhodně. V některých případech se může stát, že se i při našem zadaném počtu shluků, na výstupu tohoto algoritmu objeví počet shluků menší. Je to speciální případ. Pro ukázku tohoto případu viz *obrázek 7*.

Na tomto obrázku byly inicializovány první centroidy na pozici 1,6,7. Po první iteraci se centroidy přeskupily, jak lze vidět na levé straně obrázku. Po opětovném přepočtu má ovšem najednou bod 5 blíže k červenému shluku a rovněž původní bod 6 patřící modrému shluku se přiřadí ke shluku zelenému. Při dalším přepočtu centroidů neobsahuje modrý shluk žádné body, jeho hodnota bude nulová a již se k němu žádné body nepřiřadí.



Obrázek 7: Příklad vzniku prázdného clusteru.

Označme si I jako počet iterací, d jako počet dimenzí vektorů reprezentující dokument, k jako náš počet zvolených clusterů a n reprezentující celkový počet dokumentů pro

shlukování. Potom je časová složitost pro přiřazení do clusteru $O(nkd)$ a časová složitost výpočtu těžišť clusterů $O(nd)$. Pro I iterací je tedy celková složitost řešení:

$$O(k + I(nkd + nd)) = O(Idkn)$$

Časová složitost algoritmu je tedy nízká, ovšem za cenu možnosti výrazně se měnících výsledků.

Jak pramení z této počáteční inicializace, struktura clusterů je velmi proměnná a rovněž je zde šance, že nám může vzniknout zmiňovaný prázdný cluster. Minimalizace clusterů nám ovšem nemusí vadit. Při opětovném spuštění shlukování, při stejné sadě dokumentů, nedostaneme tedy, s největší pravděpodobností totožné clustery. Algoritmus tedy počítá lokální minima a velmi záleží na prvních centroidech. Zde je na uváženou, zda nám to moc vadí. Pokud chceme dokumenty rozřadit do nějakých kategorií, dokumentu není intuitivně striktně dána pouze jediná kategorie, ale může spadat do kategorií jiných. Pokud tedy náhodně vybíráme počáteční centroidy, přiřazujeme ostatní články k právě nejpodobnějšímu tématu vybraných dokumentů reprezentujících centroidy.

4.4.2 BuckShot inicializace

Mým nápadem, jak zlepšit přesnost algoritmu, byl vybrat náhodně, dejme tomu desetinu všech vektorů. Počet by především záležel na celkovém počtu všech dokumentů a ty všechny mezi sebou porovnávat a odstraňovat sobě nejbližší dvojice a přidávat zpět jejich průměr. Víceméně přesně tohle dělá BuckShot inicializace, kterou jsem později našel. Je to inicializace, jenž zlepšuje výkonnost algoritmu.[19]

Každý dokument je reprezentován jako samostatný cluster, při každém procházení náhodně vybraných clusterů se vypočítá vzdálenost každého clusteru s každým. Při každém propočtu se zkontroluje, zda vypočítaná vzdálenost je menší než nejmenší uložená vzdálenost, která je ze startu nastavena na nulu. Pokud je vzdálenost menší, uložím si clustery(vektory). Po skončení dostanu právě dva nejbližší clustery. Tyto dva clustery vymažu ze seznamu náhodně vybraných vektorů a vložím vektor, který reprezentuje průměr těchto dvou clusterů. Při každé iteraci se tedy zmenšuje seznam clusterů o jeden. Algoritmus končí ve chvíli, kdy je seznam clusterů roven počtu námi zvolených centroidů, které se použijí jako výchozí centroidy pro samotný k-means.

Jedná se vlastně o kombinaci HAC a klasického k-means. Jen s tím rozdílem, že si nepotřebuji pamatovat počet vektorů dokumentů v jednotlivých clusterech, ale pro naši potřebu nám stačí znát jen jeho průměr. Tedy daný cluster je roven centroidu. Počet náhodně vybraných vektorů je dán $\sqrt{n}$, kde n je počet celkového množství dokumentů, pro které se má provést shlukovací analýza.

Pokud si opět označíme n jako celkový počet dokumentů pro shlukování, k jako počet námi voleného počtu clusterů, d jako počet dimenzí ve vektoru dokumentů, tak je časová složitost algoritmu pro inicializaci centroidů následující:

$$O(\sqrt{k}n + (\sqrt{k}n - k)knd) = O((kn)^{3/2}d)$$

Celková složitost k-meas při použití tohoto inicializačního algoritmu, kde I značí počet iterací tedy je:

$$O((kn)^{3/2}d)Idkn$$

Časová složitost je tedy větší, záleží tedy dost na velikosti testovací sady dokumentů. Avšak v porovnání výsledku a času, který je potřebný navíc pro řešení s náhodnou inicializací, se tato inicializace vyplatí. V mé testovací sadě přibližně o 2200 dokumentech a vektorech o 1200 složkách je celkový potřebný čas pro shlukování navýšen přibližně na dvojnásobek.

4.4.3 Úplná HAC inicializace

Dalším mým nápadem pro inicializaci počátečních centroidů bylo zkusit použít HAC na celou sadu článků. Výsledkem by teoreticky měly být nejlépe inicializované počáteční centroidy, jejichž vnitřní podobnost by měla být větš,í než v přecházejících případech. Tato metoda se ovšem nedá moc použít pro větší sady dokumentů, kvůli její vysoké časové náročnosti. Výhodou je, že nemusíme opakovat shlukování pro statistiku průměrných výsledků, jak plyne z HAC, výsledek je vždy stejný. V této inicializaci se při každé iteraci rovněž snižuje počet vektorů v seznamu, s nímž algoritmus pracuje o jeden dokument. Seznam vektorů k porovnávání je zde mnohem větší a opět musí být každý vektor porovnán s každým. Na konci jednoho cyklu jsou právě dva vektory smazány a následně přidán vektor jejich průměru. Poté se cyklus opakuje.

Po krátkém zamyšlení jsem zjistil, že je značně nevýhodné po prvním cyklu opět přepočítávat všechny vzdálenosti. Přidávám pouze jeden vektor a ten bych měl přepočítávat s ostatními. Pro tuto optimalizaci jsem si vytvořil slovník. Při první iteraci vypočtu všechny vzájemné vzdálenosti vektoru a přidám je do mnou vytvořeného slovníku. Klíč je zde tvořen ze dvou identifikačních čísel vektorů např. 653x8963. První číslo v klíči musí být menší než číslo druhé, kvůli jednotnému formátu klíčů. Při počítání vzdálenosti mezi dvěma vektory je výsledná podobnost určitě rovna podobnosti, i když pořadí těchto vektoru zaměním.

Při smazání dvou nejbližších vektorů musím všechny záznamy ve slovníku, kde se nachází jedno či druhé identifikační číslo smazat. To provádím skládáním klíčů z těchto vektorů ze všemi identifikačními čísly vektorů v seznamu. Klíč tedy vždy skládám ve výše zmiňovaném pořadí (menší x větší) a mám vždy jistotu, že se při mazání záznamu podobnosti této dvojice, klíč ve slovníku nachází. Nemusím tedy zkoušet první či druhou klíčovou kombinaci, v případě, že bych si klíč v tomto pořadí neukládal.

Po smazání záznamu podobnosti vektorů ve slovníku, zde přidávám všechny výsledky podobnosti s ostatními vektory v seznamu. Jako identifikační číslo nového vektoru zde volím menší identifikační číslo z dvojice vektorů k vyřazení. Máme pak jistotu, že se nebude vyskytovat duplicitní identifikátor. Pokud jsou všechny propočty přidány, slovník musím seřadit dle jeho výsledku podobnosti sestupně. Klíč prvního záznamu rozložím na dva identifikátory a vektory s těmito čísly v seznamu odeberu. Poté proces odebírání, přidávání a seřazování slovníku opakuji do té doby, dokud počet vektorů v seznamu není roven počtu námi zadaných clusterů.

Při této počáteční inicializaci centroidů, při vyšším počtu clusterů, jsem dostal velmi dobré výsledky. Při malém počtu clusterů se nám může stát, že se v sadě vyskytuje pár dokumentů ve shluku velmi vzdálených od hlavního shluku. Výsledek shlukování nám pak může vyjít takový, že v jednom, dvou,třech či dokonce čtyřech clusterech z pěti budeme mít jen např. 30 dokumentů a v posledním pátém např. 800 dokumentů. Je tedy třeba zvážit, zda takový výsledek je pro nás dostačující i přes velmi dobré shlukování prvních 4 clusterů. Při větším počtu clusterů, např. 30 je případný velký shluk eliminován.

4.5 Možnosti aplikace

V této podkapitole se budu věnovat stručně možnostem, jenž má aplikace nabízí.

4.5.1 Výběr dokumentů k analýze

Jako standardní vstup se nabízí načtení textového souboru, ze kterého se načítají dokumenty. Tento soubor musí být v již zmiňovaném formátu. Pokud máme již zpracované dokumenty ve slovnících, můžeme si všechny tyto slovníky uložit a později načíst. Rychlost zpracování souboru je ovšem při menší textové sadě vysoká, proto možnost ukládání použijeme především pro uložení slovníku z celé zpracované sady dokumentů. Pokud si takto slovníky uložíme, můžeme si ručně překopírovat právě dva slovníky pro další možnosti načítání dokumentů.

Mám na mysli slovníky, kde první z této dvojice ukládá všechny dokumenty (dokumentsText) a slovník, který nám říká, v jakých konkrétních dokumentech se určitý kořen slova ze seznamu nachází (documentsWithWord). Překopírovaní slovníku provadíme především k předejití přeuložení slovníku z celé zpracované sady dokumentů a musíme je uložit do podsložky save v adresáři, kde se nachází spustitelná aplikace. Díky tomuto se nám nabízí další možnost výběru dokumentů ke shlukování. Konkrétně tedy vybrání dokumentů, kde se nachází jedno či více námi vybraných slov ze seznamu. Poslední možností je, vybrat si náhodný počet dokumentů z celé předzpracované sady.

4.5.2 Nastavení analýzy

V menu můžeme v položce setting nastavit dolní frekvenční mez, která nám určuje minimální frekvenci vyskytnutých slov, použitých pro reprezentaci složek vektorů, reprezentující dokumenty. Dále můžeme nastavit počet clusterů a zvolit inicializační metodu pro nastavení inicializačních centroidů. Po zpracování dokumentů uvidíme frekvenční analýzu všech vyskytnutých slov nad danou mezí. Vidíme, jak slova v nepřevedeném tvaru, tak i slova převedená na slovní kořen. U libovolného kořenu slova máme možnost k nahlédnutí statistiky frekvenčního výskytu nepřevedených slov o určitém stejném kořenu.

K dispozici je i graf pro lepší představu, kde vidíme, které tvary jsou nejvíce zastoupené. Dále jsou k nahlédnutí i slova, která se vyskytla pod mezí. Díky tomuto si můžeme ověřit, zda jsme si dolní mez nastavili dobře. Pokud se nám zdá, že ne, nastavíme mez jinak a poté v položce setting zvolíme Reload Documents. V aplikaci je možné upravovat

seznam stop slov. K této úpravě použijeme nabídku setting a po volbě přidání či odebrání stop slov se nám zobrazí seznam slov frekvenční analýzy nad celou sadou článku a aktuálně používaný seznam stop slov. Pokud hodláme přidat stop slovo do seznamu z frekvenční analýzy, přidají se všechny tvary vybraného kořene slova.

4.5.3 Zobrazení výsledků shlukování

Po dokončení procesu shlukování se zobrazí pane,l kde jsou zobrazeny výsledné clustery a počet dokumentů v jednotlivých clusterech. Po kliknutí na libovolný cluster je nám zobrazen formulář pro nahlížení všech dokumentů v clusterech. Slouží tedy pro viditelnou kontrolu shlukování.

K dispozici máme v nabídce rovněž zobrazení frekvenční analýzy všech clusterů. V prvotním panelu jsou opět zobrazeny všechny clustery a počty dokumentů jako v nabídce předešlé. Ovšem s tím rozdílem, že je zde ke všem clusterům přidáno prvních 15 slov z jejich celkové frekvenční analýzy. Po kliknutí na libovolný cluster v této nabídce se nám zobrazí formulář, jenž tvoří graf a bližší nabídka. V grafu je zobrazena frekvenční analýza clusteru. Pro možnost lepšího zobrazení si zde lze zvolit ze tří typů grafu či přepnout na zobrazení jiného počtu prvních slov frekvenční analýzy. V tomto formuláři se lze zajisté mezi clustery přesouvat.

Pro porovnávání výsledků frekvenční analýzy různých clusterů si zde můžeme zobrazit druhý graf, který zobrazuje výsledky frekvenční analýzy jiných clusterů. Velmi zajímavou možností je zobrazení si druhého grafu, který nám bude reprezentovat frekvenční analýzu $\sqrt{n}$ nejbližších vektorů k centroidu clusteru zobrazeného v grafu prvním, kde n je celkový počet vektoru dokumentu v daném clusteru. Díky této možnosti porovnání můžeme sami vidět, jak moc různorodý cluster je. Hlavním důvodem implementace této frekvenční analýzy bylo splnit téma této bakalářské práce, což je detekce kllíčových slov.

Další a poslední nabídkou je zobrazení panelu s vnitřní podobností jednotlivých clusterů a jejich průměrná podobnost. Opakování shlukování, např. pod jinou inicializační metodou, či jiným počátečním počtem clusterů nad stejnou sadou dokumentů nalezneme v menu v nabídce setting.

5 Experimenty

V této sekci se budu věnovat mnou zvolenými experimenty nad výsledkem procesu shlukování. Tento proces sloužil jako nástroj pro určitou kategorizaci, nad jejímiž výsledky budeme aplikovat frekvenční analýzu a snažit se detekovat charakteristická klíčová slova pro jednotlivé shluky.

5.1 Vnitřní podobnost clusterů

Tento experiment se netýká samotné detekce klíčových slov, nýbrž vnitřní podobnosti shluků při použití všech třech inicializačních metod. Vnitřní podobnost se vypočítá jako průměr podobností každého vektoru v clusteru s jeho centroidem.

Pro přesnější výsledky jsem při náhodné a buckshot inicializaci meření opakoval 5-krát, výsledná průměrná hodnota byla zapsána do tabulky. Inicializační metodu s kompletní hac inicializací stačí použít pouze jednou, protože vrací vždy stejný výsledek nad danou sadou dokumentů. Výsledné hodnoty měření zobrazuje *tabulka 7*, kde byla zvolena spodní frekvenční mez četnosti slova pro započtení do složek vektoru na 10. Pro druhé měření byla použita mez = 30, jejíž výsledky zobrazuje *tabulka 8*.

metoda/počet dok	500	1000	1500	2000	3000
RANDOM	0.2271	0.1978	0.1923	0.185	0.1747
BUCKSHOT	0.2439	0.227	0.209	0.204	0.1824
HAC	0.2704	0.213	0.208	0.211	0.1851

Tabulka 7: Vnitřní podobnost clusterů, mez =10

metoda/počet dok	500	1000	1500	2000	3000
RANDOM	0.3181	0.,2409	0.222	0.207	0.205
BUCKSHOT	0.3442	0.2603	0.2421	0.213	0.207
HAC	0.3321	0.2552	0.2403	0.221	0.219

Tabulka 8: Vnitřní podobnost clusterů, mez =30

Jak lze vidět, inicializační metody mají vliv na podobnost clusterů. Malé hodnoty podobnosti si lze vysvělit široce obsáhlými textovými dokumenty, kdy centroid obsahuje skoro všechny složky. Správnost shlukování si lze ověřit použitím frekvenční analýzy a výpisem článku jednotlivých clusterů. Velký vliv na podobnost clusterů má také nastavení spodní frekvenční meze. Počet složek vektorů v druhém měření se nám redukoval v průměru o 30%.

5.2 Frekvenční analýza clusterů

Díky celkové frekvenční analýze jednotlivých clusterů můžeme posoudit různorodost clusterů objektivně podle top slov jednotlivých clusterů. Taktéž si díky těmto top slovům

můžeme říci, která slova jsou pro daný cluster klíčová. Jsou to především slova, jenž se vyskytují hojně v jednom clusteru a v jiných ne nebo se v nich vyskytují, ale s malou frekvencí. Pro upřesnění detekce těchto klíčových slov můžeme použít frekvenční analýzu nejbližších dokumentů k centroidu shluku. Je rovněž důležité, do analýzy nezapočítávat slova, která se vyskytují s vysokou frekvencí, protože se nám může stát, že se nám místo klíčových slov ve frekvenční analýze zobrazí obecně častá slova.

5.3 Dokumenty s danými klíčovými slovy

Pro tento experiment si z celkové sady 530 000 článků nechám vyhledat pouze články, kde se vyskytují slova **world**, **wide** a **web**. Tato slova se alespoň jednou samostatně vyskytují v následujícím počtu dokumentů:

- **world**: 24 232 dokumentů
- **wide**: 31 222 dokumentů
- **wide**: 33 527 dokumentů

Průnikem množin těchto dokumentů dostaneme 2 254 dokumentů pro shlukování a necháme si je roztřídit do 6 kategorií při nastavení spodní meze na hodnotu 15. Použijeme inicializační metodu BuckShot. Pro náš experiment je důležité, vyřadit obecně častá slova z frekvenční analýzy jednotlivých clusterů. Která slova to budou, nám může říct frekvenční analýza nad celou sadou 530 000 dokumentů, kterou již mám předzpracovanou a uloženou.*Tabulka 10* zobrazuje prvních 10 slov s největším výskytem slov v dokumentech, kde se nachází slova wide, world, web. *Tabulka 9* slouží pro porovnání frekvenčního výskytu top slov *tabulky 9* napříc celou sadou 530 000 dokumentů.

index	slovo	frekvence
61	web	91 865
21	inform	185 174
>100	world	31 085
>100	wide	34 939
26	user	157 578
3	paper	341 309
34	servic	129 342
87	search	74 373
17	applic	207 040
25	provid	163 033

Tabulka 9: Top 10 slov v kořenovém tvaru ve frekvenční analýze celé sady 530 000 dokumentů.

index	slovo	frekvence
1	web	7 536
2	inform	2 898
3	world	2 807
4	wide	2 736
5	user	2 344
6	paper	1 609
7	servic	1 526
8	search	1 317
9	applic	1 314
10	provid	1 254

Tabulka 10: Top 10 slov ve frekvenční analýze dokumentů, kde se vyskytují slova world, wide a web.

Jak lze vyčíst z obou tabulek výše, vyskytuje se nám v naší sadě dokumentů 6 slov s frekvenčním výskytem nad 100 000. Nebudou s velkou pravděpodobností představovat

klíčová slova těchto shluků, protože jako složka ve vektoru, by měly nastaveny malou váhu (číslo). Pro mou detekci klíčových slov si tedy dovolím odstranit z frekvenční analýzy slova, jejichž výskyt napříč celou sadou 530 000 dokumentů, je vyšší jak 100 000. Frekvenční analýzu nad jednolivými clustery shlukování lze vidět v *tabulce 11*.

Pro lepší objektivitu při hledání klíčových slov jsou v *tabulce 12* zobrazeny výsledky frekvenční analýzy nebližších dokumentů k centroidu shluku. Počet těchto nejbližších dokumentů je roven $\sqrt{n}$, kde n je počet dokumentů v clusteru. Počty dokumentů v clusterech se pohybují v rozmezí od 100-300 dokumentů až na výjimku clusteru 5, který reprezentuje 970 dokumentů.

index	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6
1	web	web	web	web	web	web
2	ontlog	world	world	server	world	world
3	semant	wide	wide	wide	wide	page
4	world	mobil	agent	world	search	wide
5	wide	internet	technolog	cach	queri	grapf
6	visual	technolog	mine	client	document	social
7	map	access	internet	access	languag	cluster
8	concept	devic	access	traffic	engin	commun
9	knowledg	commun	virtual	protocol	page	link
10	languag	content	semant	internet	xml	content

Tabulka 11: Frekvenční analýza top 10 nejvyskytovanějších slov v jednotlivých clusterech.

index	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6
1	ontolog	mobil	agent	cach	queri	graph
2	web	web	web	proxi	search	web
3	semant	devic	intellig	web	web	commun
4	integr	access	wide	server	engin	node
5	rule	technolog	world	world	xml	cluster
6	wide	wireless	articl	wide	retriev	edg
7	languag	internet	semant	dynam	document	world
8	domain	contect	retriev	reduc	databas	wide
9	world	offer	sourc	internet	languag	page
10	engin	ip	issu	request	wide	random

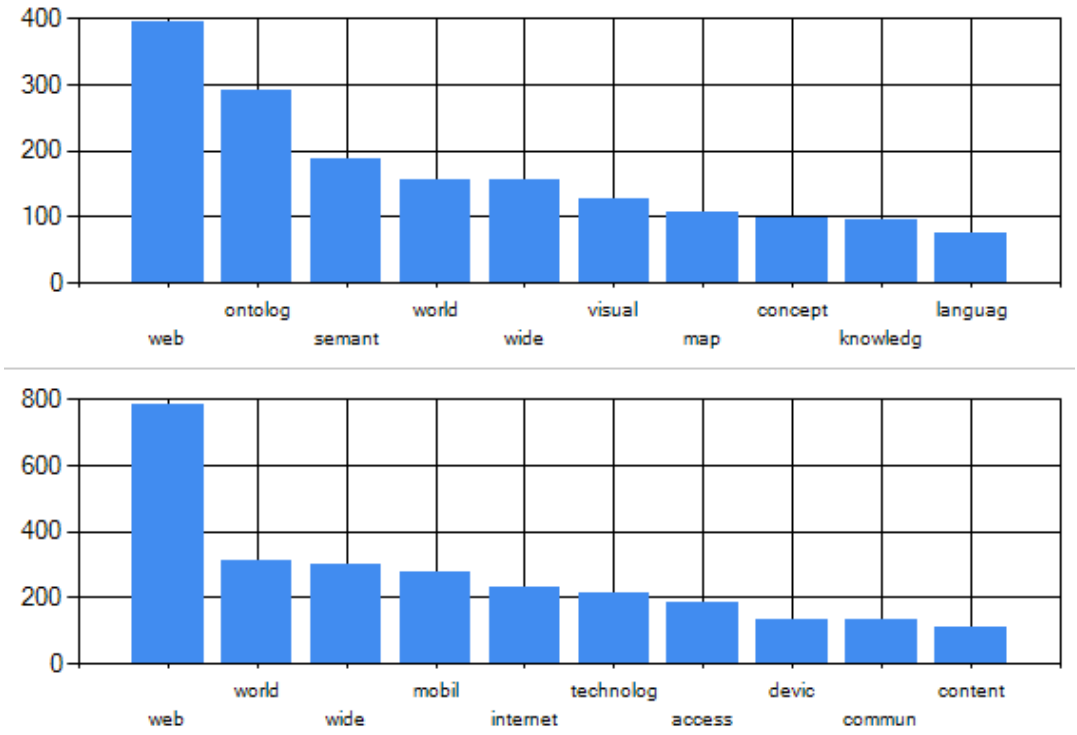
Tabulka 12: Frekvenční analýza top 10 nejvyskytovanějších slov v nejbližších dokumentech centroidů clusterů.

Z výše uvedených tabulek lze tedy říci, že pro jednotlivé clustery tvoří slova world, wide a web kontext (souvislost) nad všemi clustery a jednotlivé subkontexty v clusterech tvoří právě klíčová slova, jenž jsem detekoval frekveční analýzou. Klíčová slova v jednotlivých clusterech jsem vybral především na základě jedinečnosti výskytu v clusteru. Viz *tabulka 13.*

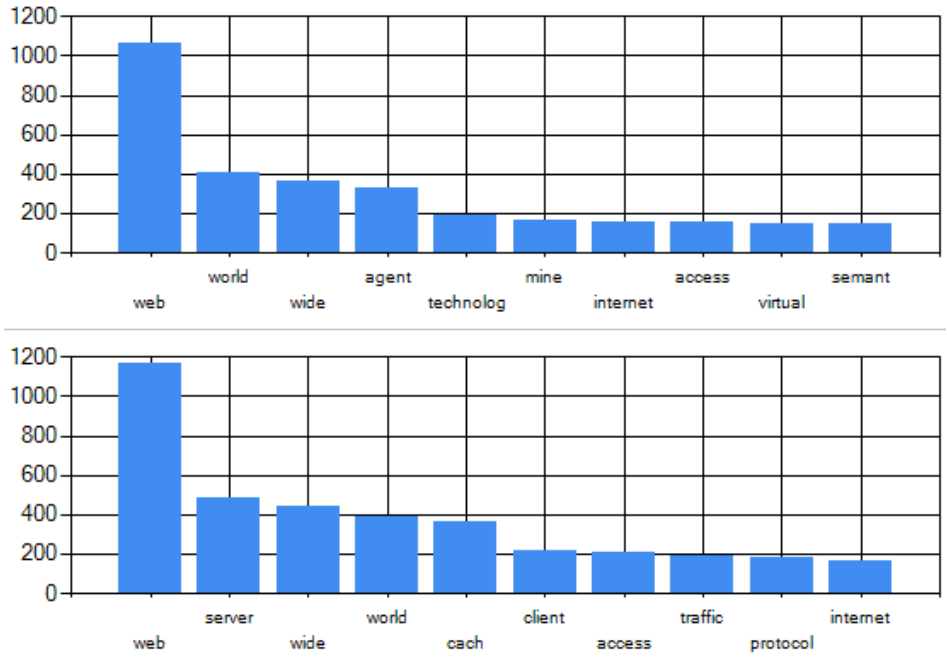
index	1	2	3	4	
cluster 1	ontolog	semant	visual	map	concept
cluster 2	mobil	internet	technolog	access	devic
cluster 3	agent	technolog	mine	internet	access
cluster 4	server	cach	client	access	traffic
cluster 5	search	queri	document	engin	languag
cluster 6	graph	page	social	cluster	commun

Tabulka 13: Výpis jednotlivých klíčových slov clusterů.

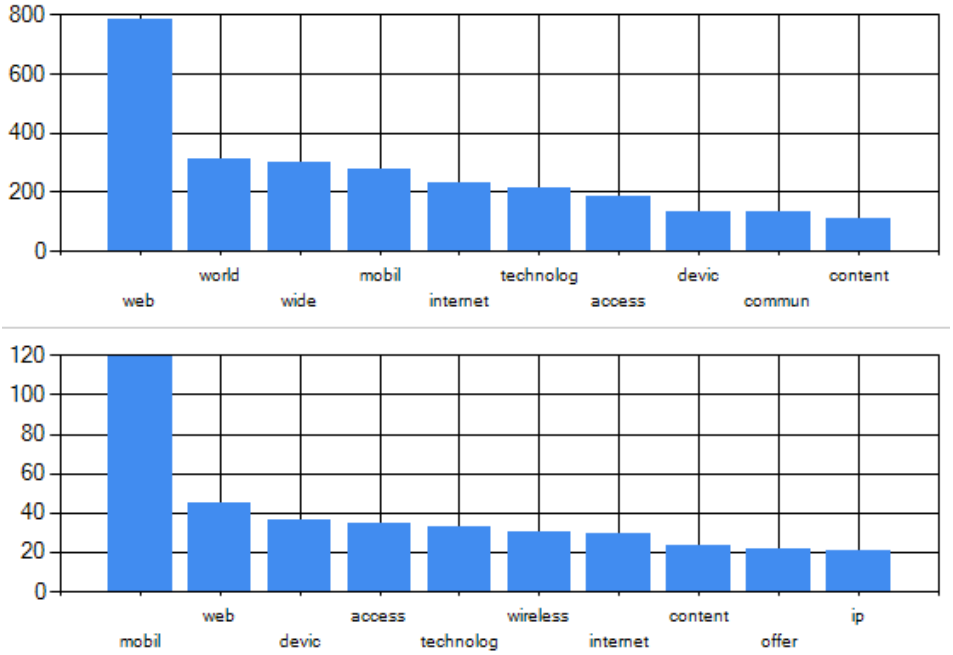
Z výpisu detekovaných klíčových slov z*tabulky 13*, kde se skoro v žádném nevyskytla slova nejedinečná, mohu tedy usoudit, že jsem klíčová slova detekoval správně. Tato detekováná klíčová slova by s velkou pravděpodobností mohla reprezentovat určitou kategorii pro daný cluster. Tato sada klíčových slov by mohla sloužit např. jako dokument v trénovací sadě dat, kdy by moje řešení reprezentovalo případného znalce pro posouzení kategorie.
Příklad grafického znázornění různorodosti klíčových slov frekvenční analýzy clusteru 1,2 a 3,4 v mé aplikaci, lze vidět na *obrázku 8* a *obrázku 9*. Frekvenční analýzu celého clusteru 2 a frekvenční analýzu jeho dokumentů nejblíže k centroidu lze pak vidět na *obrázku 10*.



Obrázek 8: Grafický výstup mé aplikace pro znázornění frekvenční analýzy clusteru 1,2.



Obrázek 9: Grafický výstup mé aplikace pro znázornění frekvenční analýzy clusteru 3,4.



Obrázek 10: Grafický výstup mé aplikace pro znázornění frekvenční analýzy clusteru 1 a frekvenční analýzy jeho nejbližších dokumentů

5.4 Detekce klíčových slov v náhodně vybraných dokumentech.

V této sekci již nebudu rozebírat frekvenční analýzu nad celou sadou dokumentů určenou ke shlukování. Budu postupovat stejnými postupy jako v předcházejícím případě, zde si ovšem vyberu sadu 3000 dokumentů náhodně. Spodní mez frekvenčního výskytu slova potřebnou pro započtení do slova jako složky vektoru je stanovena na 15 a počet shluků na číslo 6 jako v předchozím případě.

index	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6
1	protocol	language	web	fuzzi	graph	architectur
2	rout	agent	web	motion	bound	environ
3	node	queri	task	fault	case	technolog
4	mobil	formal	semant	featur	linear	interact
5	secur	specif	search	estim	tree	parallel
6	wireless	orient	ontolog	human	complex	commun
7	traffic	knowledg	retriev	recognit	point	manag
8	scheme	semant	queri	robust	given	featur
9	packet	express	group	parametr	approxim	real
10	commun	tool	research	signal	space	research

Tabulka 14: Frekvenční analýza top 10 nejvyskytovanějších slov clusterů při náhodně zvolených dokumentech.

index	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6
1	rout	language	web	fuzzi	graph	parallel
2	node	agent	semant	human	bound	schedul
3	protocol	queri	document	approxim	case	architectur
4	mobil	orient	text	rule	degre	cluster
5	wireless	semant	content	fault	edg	commun
6	adhoc	formal	retriev	classif	class	processor
7	senzor	knowledg	search	weight	vertex	environ
8	manag	framework	larg	estim	match	real
9	link	express	engin	recognit	properti	power
10	secur	theori	measur	motion	vertic	execute

Tabulka 15: Frekvenční analýza top 10 nejvyskytovanějších slov v nejbližších dokumentech k centroidů daných clusterů při náhodně zvolených dokumentech.

Z výše uvedených tabulek lze říci, že pro všechny clustery zde nemáme žádný jednotný kontext a jednotlivé subkontexty v clusterech tvoří právě klíčová slova, které jsem detekoval frekveční analýzou. Různorodost náhodně vybraných dokumentů se nám projeví také právě na klíčových slovech, která již nebudou tak specifická, ale spíše obecnějšího rázu. viz *tabulka 16.*

index	1	2	3	4	5
cluster 1	rout	node	protocol	wireless	packet
cluster 2	languag	agent	queri	specif	knowledg
cluster 3	web	document	search	ontolog	semant
cluster 4	fuzzi	motion	fault	featur	estim
cluster 5	graph	bound	linear	tree	complex
cluster 6	architecture	parallel	technolog	interact	schedul

Tabulka 16: Výpis jednotlivých klíčových slov clusterů při náhodně vybraných dokumentech

6 Závěr

Při hledání daných metod, mne zaujal nehierarchický algoritmus k-means, a to pro své velké využití v praxi. Rozhodl jsem se tedy, že se pokusím o tomto algoritmu dozvědět více a zjistit, jak by jej šlo využít pro mé téma. Tento algoritmus mi při použití kosinové podobnosti posloužil jako kvalitní nástroj pro kategorizaci dokumentů, na základě jejich podobnosti. Nad výsledkem shlukování tohoto algoritmu, při odfiltrování obecně častých slov, jsem pak již mohl lehce použít frekvenční analýzu početního výskytu slov v daných clusterech a poznat jaká slova jsou pro tento cluster charakteristická. Detekoval jsem tedy právě klíčová slova pro dokumenty, které se vyskytly v tomto clusteru.

Při jednotlivých krocích ke splnění zadaného tématu mým řešením a nabývání praktických zkušeností ve vědecké oblasti dolování textu, jsem si dával dohromady souvislosti, které mi byly dříve nejasné či naprosto nepochopitelné. Největší přínos této práce byl tedy pro mne v získání velmi dobrých praktických zkušeností. Dalším přínosem bylo, že jsem u zadané sady dokumentů dokázal detekovat klíčová slova pomocí nehierarchického algoritmu k-means, nad jednotlivými subsadami celkové sady dokumentů. Tato klíčová slova skupiny dokumentů, která jsou podobná, mohou být použita např. pro trénovací sadu dokumentů jiných algoritmů.

Díky nabytým zkušenostem bych řadu věcí v aplikaci vylepšil. Jedná se především o složitější rozsahy inicializačních metod prvních centroidů clusterů. Určitě bych také implementoval nástavbu pro automatickou detekci klíčových slov. Jelikož je mi oblast, kterou se má práce zabývala mnohem bližší a o danou problematiku jsem se začal zajímat, uvažuji o tom, že bych na toto téma navázal později i v práci diplomové. Pokud by se tak nestalo, mohla by se má další práce týkat oblasti dolování textu či dolování dat.

Na základě výsledku mé práce si mohu dovolit konstatovat, že jsem stanovený cíl práce splnil.

Ondřej Blažek

7 Reference

[1] maths.cz *"Euklidovská vzdálenost"*, [ONLINE], http://maths.cz/clanky/analyticka-geometrie-skalarni-soucin.html

[2] fulltext.sblog.cz *"Pravděpodobnostní sématická latentní analýza"*, [ONLINE], http://fulltext.sblog.cz/2011/12/04/semanticka-analyza-textu-5/

[3] nlp.stanford.ed *"Stemming a lematizace"*, [ONLINE], http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[4] nlp.stanford.edu *"Hierarchický aglomerativní clustering"*, [ONLINE], http://nlp.stanford.edu/IR-book/html/htmledition/hierarchical-agglomerative-clustering-1.html

[5] nlp.stanford.edu *"Divizive clustering"*, [ONLINE], http://nlp.stanford.edu/IR-book/html/htmledition/divisive-clustering-1.html

[6] fulltext.sblog.cz *"Reprezentace vector space model"*, [ONLINE], http://fulltext.sblog.cz/2011/08/30/semanticka-analyza-textu-1/

[7] cs.wikipedia.org *"TF-IDF"*, [ONLINE], http://cs.wikipedia.org/wiki/Tf-idf

[8] fulltext.sblog.cz *"Latentní sémantická analýza"*, [ONLINE], http://fulltext.sblog.cz/2011/10/17/semanticka-analyza-textu-4/

[9] semanticsearchart.com *""Latentní sémantická analýza"*, [ONLINE], http://www.semanticsearchart.com/researchLSA.html

[10] nltk.googlecode.com *"Naivní bayes klasifikátor"*, [ONLINE], http://nltk.googlecode.com/svn/trunk/doc/book/ch06.html

[11] fulltext.sblog.cz *"Metody pro výpočet dopobnosti vektorů."*, [ONLINE], http://fulltext.sblog.cz/2011/09/12/semanticka-analyza-textu-2/

[12] semanticsearchart.com *"Naivní bayes klasifikátor"*, [ONLINE], http://www.semanticsearchart.com/researchNB.html

[13] saedsayad.com *"Naivní bayes klasifikátor"*, [ONLINE], http://www.saedsayad.com/naive_bayesian.htm

[14] ise.bgu.ac.il *"Clustering metody"*, [ONLINE], http://www.ise.bgu.ac.il/faculty/liorr/hbchap15.pdf

[15] cs.princeton.edul *"Latentní dirichletova alokace"*, [ONLINE],
http://www.cs.princeton.edu/~blei/papers/BleiNgJordan2003.pdf

[16] fulltext.sblog.cz *"Latentní dirichletova alokace"*, [ONLINE],
http://fulltext.sblog.cz/2011/12/22/semanticka-analyza-textu-6/

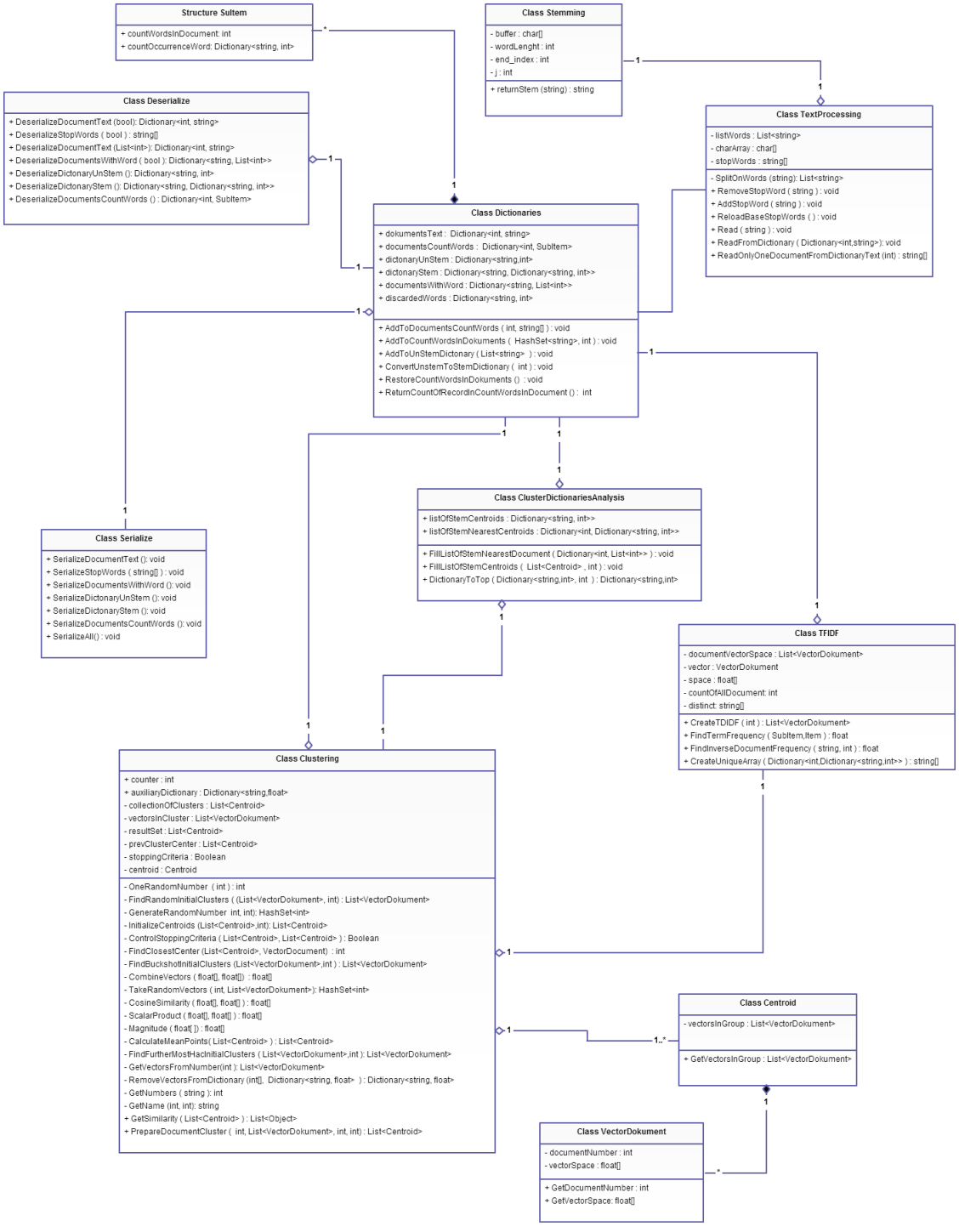
[17] bus.utk.edu *"Latentní dirichletova alokace"*, [ONLINE],
http://bus.utk.edu/stat/stat579/Hierarchical%20Clustering%
20Methods.pdf

[18] fulltext.sblog.cz *"Snižování dimenzionality"*, [ONLINE],
http://fulltext.sblog.cz/2011/09/22/semanticka-analyza-textu-3/

[19] rakaposhi.eas.asu.edu *"BuckShot algoritmus"*, [ONLINE],
http://rakaposhi.eas.asu.edu/cse494/f05-projects/
bhaumik-Project_C_Report.pdf

[20] rakaposhi.eas.asu.edu *"Sada dokumentů nad niž jsem pracoval"*, [ONLINE],
http://www.arnetminer.org/citation

A Třídní diagram



B Vývojový diagram shlukování